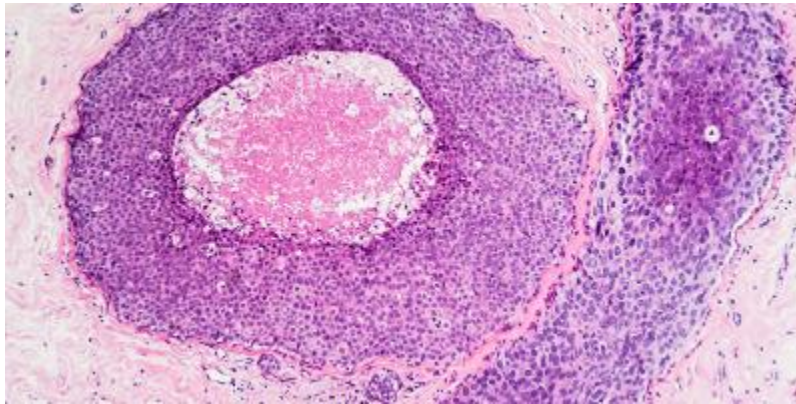


CARCINOMA

By:

Gerald Nika



Done on September 14, 2020

Carcinoma:

Abstract:

“**Carcinoma**” is a Python application, developed in Jupyter Notebook platform, built on top of the Keras framework. This software aims to help doctors classify breast tumors as benign or malignant, by using some simple statistical methods of Machine Learning. The architecture used falls under the category of Deep Learning and is called Artificial Neural Network (ANN). There is a total of three models provided in this project, one is the model which applies the Logistic Regression method, and the two others apply the Softmax Regression method. The thesis is towards research, since it conveys an experimental and analytical approach of the problem.

Firstly, I have presented the problem in terms of biology, then I explain the nature of data used throughout this project.

Then, before diving into details of the project coding, I give explanation to some concepts in Machine Learning and Statistics, which will be used in this project, and also, I give the reason why I have used these methods and not some other.

In the end of each model I draw conclusions about the learning and testing performance.

Then in the end of all the three models, I compare the models’ performances and also give some trade-offs and other possible ways how this problem could have been approached.

Keywords:

Data Science, Machine Learning, Deep Learning, Artificial Neural Network, Statistics, Logistic Regression, Softmax Regression, Activation Function, Loss Function, Invasive Ductal Carcinoma.

Abbreviations

“Data Science”	—	DS;
“Machine Learning”	—	ML;
“Artificial Neural Network”	—	ANN;
“Microsoft Cognitive Toolkit”	—	CNTK;
“Application Programming Interface”	—	API;
“Invasive Ductal Carcinoma”	—	IDC;
“Comma-Separated Values”	—	CSV;
“Rectified Linear Unit”	—	ReLU;
“Exponential Linear Unit”	—	ELU;
“Logistic Regression”	—	LR;
“Softmax Regression”	—	SR;
“Mean Squared Error”	—	MSE;
“Mean Absolute Error”	—	MAE;
“Mean Percentage Error”	—	MPE;
“Area Under the Curve”	—	AUC;
“Receiver Operating Characteristic”	—	ROC;
“True Positive”	—	TP;
“False Positive”	—	FP;
“True Negative”	—	TN;
“False Negative”	—	FN;
“True Positive Rate”	—	TPR;
“False Positive Rate”	—	FPR;
“True Negative Rate”	—	TNR;
“False Negative Rate”	—	FNR.

Table of Contents

Carcinoma:	ii
Abstract:	ii
Keywords:	ii
Abbreviations	iii
List of Figures	vi
Chapter 1	1
1.1 Overview	1
Chapter 2	4
2.1 The problem	4
2.2 Tools needed	5
2.3 Explanation of Used Concepts	6
2.3.1 Artificial Neural Networks	6
2.3.2 Activation Functions	8
2.3.3 Regression	10
2.3.4 Loss Functions	17
Chapter 3	22
3.1 Overview	22
3.1.1 matrix_of_features_x.csv file:	24
3.1.2 matrix_of_labels_y.csv file:	24
3.1.3 The Classification Model.ipynb file:	25
3.2 Steps followed to create the models:	26
3.3 Model #1: Logistic Regression Model	27
3.3.1 Importing the Libraries	27
3.3.2 Importing the Dataframes	27
3.3.3 Standardization Scaling the Features	28
3.3.4 Splitting of the Dataset	28
3.3.5 Contructing the Neural Network	29
3.3.6 Early Stopper	30
3.3.7 Loss and Accuracy of the Training Set and Validation Set	31
3.3.8 Loss and Accuracy of the Testing Set	32

3.3.9 AUC Score of Testing Set	34
3.3.10 ROC Curve of the Testing Set	35
3.3.11 AUC Score of the Training Set	35
3.3.12 ROC Curve of the Training Set	36
3.3.13 Conclusion	36
3.4 Model #2: Softmax Regression Model	37
3.4.1 Converting Integer Array to Binary Class Matrix.....	37
3.4.2 Constructing the Neural Network	39
3.4.3 Early Stopper	39
3.4.4 Loss and Accuracy of the Training Set and Validation Set	40
3.4.5 Loss and Accuracy of the Testing Set	41
3.4.6 AUC Score and ROC Curve of the Testing Set	41
3.4.7 AUC Score and ROC Curve of the Training Set	42
3.5 Comparison: Logistic Regression vs. Softmax Regression	43
3.6 Model #3: Deep Learning Softmax Regression Model	44
3.6.1 Constructing the Neural Network	44
3.6.2 Early Stopper	44
3.6.3 Loss and Accuracy of the Training Set and Validation Set	45
3.6.4 Loss and Accuracy of the Testing Set	46
3.6.5 AUC Score and ROC Curve of Testing Set.....	46
3.6.6 AUC Score and ROC Curve of Training Set	47
3.7 Conclusion	48
References	49

List of Figures

FIGURE 1. NEURAL NETWORKS BIOLOGICAL ANALOGY	1
FIGURE 2. JUPYTER NOTEBOOK PLATFORM	2
FIGURE 3. KERAS FRAMEWORK	2
FIGURE 4. PYTHON DATA SCIENCE LIBRARIES.....	3
FIGURE 5. TUMOR CLASSIFICATION ANIMATION.....	4
FIGURE 6. HISTOLOGY IMAGE	5
FIGURE 7. ARTIFICIAL NEURAL NETWORK.....	6
FIGURE 8. A SINGLE NEURON #1	7
FIGURE 9. A SINGLE NEURON #2	7
FIGURE 10. SIGMOID FUNCTION	8
FIGURE 11. TANH FUNCTION	9
FIGURE 12. RELU	9
FIGURE 13. LINEAR REGRESSION #1	10
FIGURE 14. BACK PROPAGATION	11
FIGURE 15. VANISHING GRADIENT PROBLEM	11
FIGURE 16. SIGMOID DERIVATIVE.....	12
FIGURE 17. LEAKY RELU	13
FIGURE 18. ELU	13
FIGURE 19. SOFTMAX FUNCTION.....	14
FIGURE 20. SIMPLE NEURAL NETWORK	15
FIGURE 21. LINEAR SEPERABILITY.....	15
FIGURE 22. MULTI-LAYERED NEURAL NETWORK.....	16
FIGURE 23. LINEAR REGRESSION #2	17
FIGURE 24. MEAN PERCENTAGE ERROR	18
FIGURE 25. OUTLIERS EFFECT.....	19
FIGURE 26. MEAN ABSOLUTE ERROR	19
FIGURE 27. HUBER LOSS VS. SMOOTH MAE AND PREDICTED VALUES.....	20
FIGURE 28. LOGARITHMIC LOSS.....	21
FIGURE 29. ANACONDA NAVIGATOR	22
FIGURE 30. JUPYTER NOTEBOOK ENVIRONMENT.....	23
FIGURE 31. PROJECT DIRECTORY ON JUPYTER NOTEBOOK ENVIRONMENT	23
FIGURE 32. MATRIX OF FEATURES X	24
FIGURE 33. MATRIX OF LABELS Y	24
FIGURE 34. CLASSIFICATION MODEL FILE	25
FIGURE 35. IMPORTING THE LIBRARIES.....	27
FIGURE 36. IMPORTING THE DATAFRAMES.....	27
FIGURE 37. STANDARDIZATION SCALING THE FEATURES.....	28
FIGURE 38. SPLITTING THE DATASET	28
FIGURE 39. DATASET SPLITTING SCHEMA.....	28

FIGURE 40. M#1: CONSTRUCTING THE NEURAL NETWORK	29
FIGURE 41. M#1: EARLY STOPPER.....	30
FIGURE 42. M#1: LOSS AND ACCURACY OF THE TRAINING SET AND VALIDATION SET	31
FIGURE 43. M#1: LOSS AND ACCURACY OF THE TESTING SET.....	32
FIGURE 44. CASES SCHEMA	32
FIGURE 45 M#1: AUC SCORE OF TESTING SET	34
FIGURE 46. M#1: ROC CURVE OF THE TESTING SET	35
FIGURE 47. M#1: AUC SCORE OF THE TRAINING SET.....	35
FIGURE 48 M#1: ROC CURVE OF THE TRAINING SET	36
FIGURE 49. M#2: CONVERTING INTEGER ARRAY TO BINARY CLASS MATRIX #1	37
FIGURE 50. M#2: CONVERTING INTEGER ARRAY TO BINARY CLASS MATRIX #2	38
FIGURE 51. MULTI-CLASS VS. MULTI-LABEL SCHEMA.....	38
FIGURE 52. M#2: CONSTRUCTING THE NEURAL NETWORK	39
FIGURE 53. M#2: EARLY STOPPER.....	39
FIGURE 54. M#2: LOSS AND ACCURACY OF THE TRAINING SET AND VALIDATION SET	40
FIGURE 55. M#2: LOSS AND ACCURACY OF THE TESTING SET.....	41
FIGURE 56. M#2: AUC SCORE OF THE TESTING SET.....	41
FIGURE 57. M#2: ROC CURVE OF THE TESTING SET	41
FIGURE 58. M#2: AUC SCORE OF THE TRAINING SET.....	42
FIGURE 59. M32: PLOTTING THE ROC CURVE OF THE TRAINING SET.....	42
FIGURE 60. M#3: CONSTRUCTING THE NEURAL NETWORK	44
FIGURE 61. M#3: EARLY STOPPER.....	44
FIGURE 62. M#3: LOSS AND ACCURACY OF THE TRAINING SET AND VALIDATION SET	45
FIGURE 63. M#3: LOSS AND ACCURACY OF THE TESTING SET.....	46
FIGURE 64. M#3: AUC SCORE OF TESTING SET.....	46
FIGURE 65. M#3: ROC CURVE OF TESTING SET	46
FIGURE 66. M#3: AUC SCORE OF TRAINING SET.....	47
FIGURE 67. M#3: ROC CURVE OF TRAINING SET	47

Chapter 1

1.1 Overview

This software offers the ability to classify whether a tumor in a woman's breast is benign or malignant. This classifying process is made possible by using Machine Learning concepts, specifically Deep Learning concepts, such as Artificial Neural Networks (ANN).

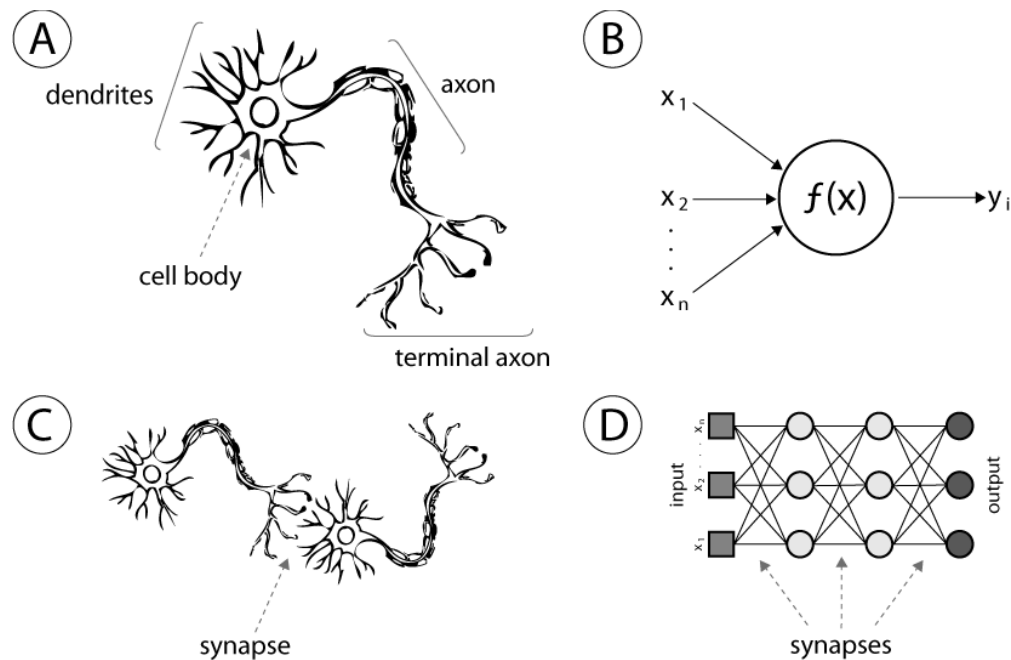


Figure 1. Neural Networks Biological Analogy

Artificial Neural Networks (ANN) are computer systems, or architectures, inspired by the workings of the biological human neural networks. An ANN, by analogy, is made up of a group of nodes called neurons, connected together by synapses, called weights, which are placed in multiple layers and produce outputs from the given inputs.

This project is developed on Jupyter Notebook environment, which makes it possible to execute blocks of code step by step and having the ability to import any library you need for your project.

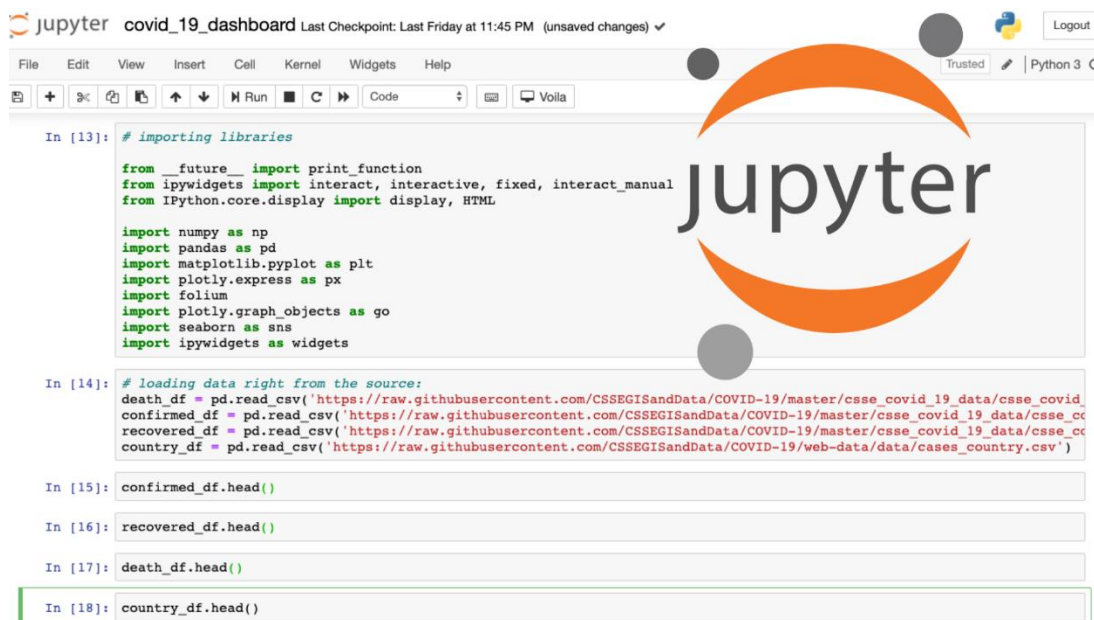


Figure 2. Jupyter Notebook Platform

The framework in which I wrote the Python code is Keras. Keras is an open-source Neural Network library written in Python and which works on top of Tensorflow, CNTK and Theano. It is a high-level API and it makes it easier for data scientists to run their new experiments. It is deployable almost everywhere and it is a vast ecosystem which covers every step of Machine Learning workflow.

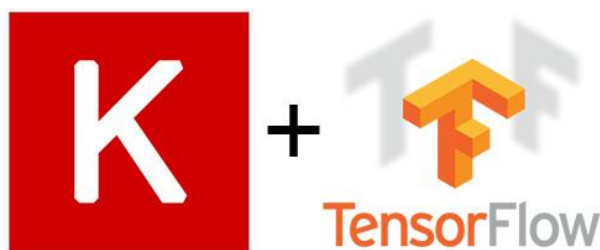


Figure 3. Keras Framework

Some of the libraries imported besides **Tensorflow** and **Keras**, are **Numpy** which deals with arrays, matrices and other mathematical functions to operate on arrays; **Pandas** which offers data structures and operations to manipulate numerical tables and time series; **Scikit-Learn** which features various classification, regression and clustering algorithms including support vector machines; and **Matplotlib** which provides an object-oriented API for embedding plots into applications.



Figure 4. Python Data Science Libraries

Chapter 2 provides a full explanation of the methods used to develop this software. First, I will cover the problem shortly, then explain the tools needed for the solution. Afterwards I will explain how these tools work underneath the hood, and then I will continue with the source code. After all this is finished, I will draw conclusions about my results and give some trade-offs about other possible ways this problem could have been approached.

I will be including images from the code execution, such as outputs and graph plot.

Chapter 2

2.1 The problem

Breast Cancer is cancer that forms in the cells of breast. It is the most common form of cancer diagnosed in both the women and men, but most cases are discovered in women. Cancers are also called bad tumors.

A tumor is a swelling part of the body, generally without inflammation, caused by abnormal growth of tissue. They can be classified as benign or malignant.

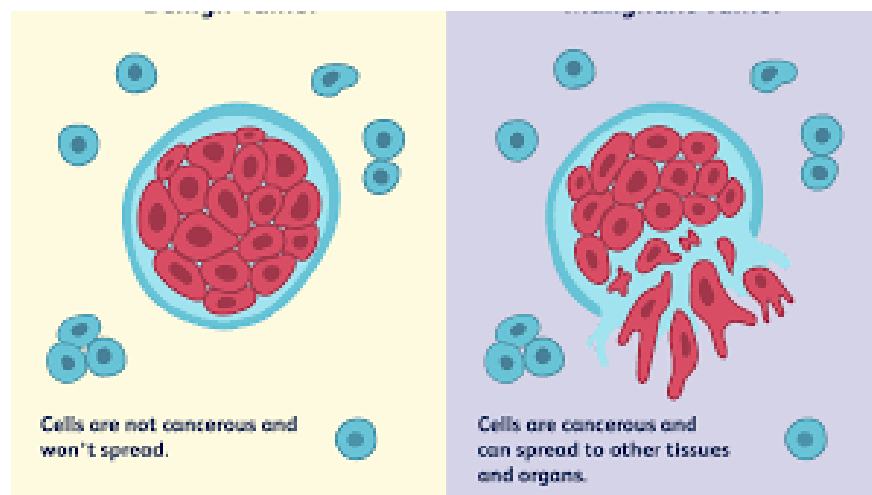


Figure 5. Tumor Classification Animation

A benign tumor is a group of cells which doesn't have the ability to invade the neighboring tissue or metastasize (spread through the body).

Malignant tumors are cancerous. They are also called IDC (Invasive Ductal Carcinoma). They develop in a milk duct and breach the fibrous or fatty breast tissue. That's when they start to metastasize and become life-threatening.

The way these two can be visually identified is through histology images. Histology is the study of the microscopic structure of tissues.

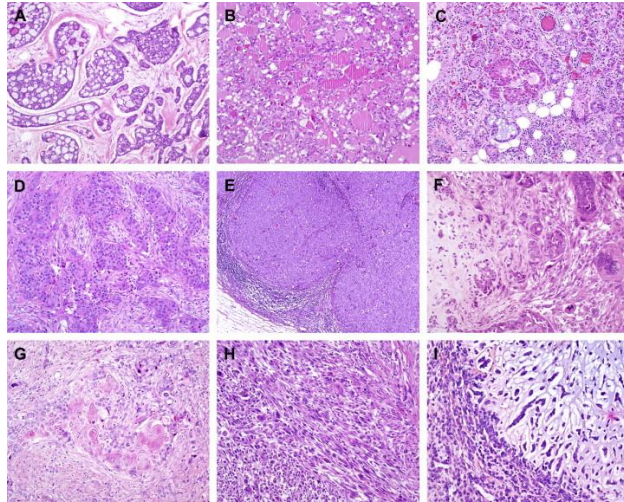


Figure 6. Histology Image

2.2 Tools needed

To be able to have our model trained, so that it can know these tumors well, we will need real histology data from previous diagnoses. This dataset can be found in UCI Machine Learning Repository.

UCI Machine Learning Repository website:

<https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>

This dataset contains 31 parameters, of which 30 of them are input features and one is the classification digit. The input features are some measurements of the nuclei of cells, such as **radius** and **texture**, **perimeter** and **area**, **smoothness** and **compactness**, **concavity** and **concave points**, **symmetry** and **fractal dimension**.

It has a total of 569 instances and has no missing values.

There are two CSV files which make up the whole dataset. One is the **matrix_of_features_x.csv**, which has the input features I mentioned, and one is the **matrix_of_labels_y.csv** file which contains all the categorical values labeled 1 or 0, for simplicity reasons.

1 is for **malignant tumor** and **0** is for **benign tumor**.

2.3 Explanation of Used Concepts

2.3.1 Artificial Neural Networks

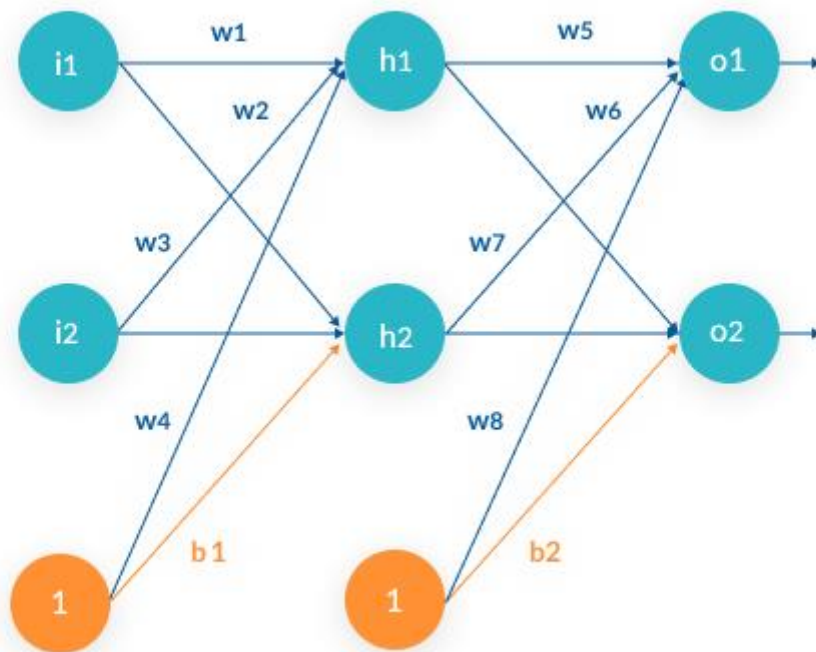


Figure 7. Artificial Neural Network

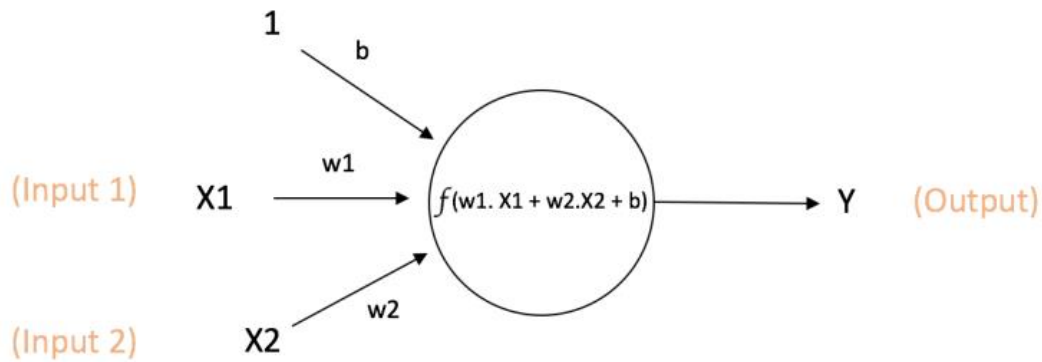
This is an example of an Artificial Neural Network (ANN) architecture.

ANN is a computer system or model composed of at least three layers. The **input** layer, the **hidden** layer and the **output** layer.

The input layer contains all the inputs. In our case these inputs are all the features in the matrix of features x .

Then you can define as many hidden layers as you want, with as many neurons inside of them as you want.

And lastly the output layer contains one or more nodes, the values of which yield a discrete probability distribution.



$$\text{Output of neuron} = Y = f(w1.X1 + w2.X2 + b)$$

Figure 8. A Single Neuron #1

A **neuron** is the most fundamental part of an ANN. It is also called a **node**. Its duty is to receive input from the predecessor nodes and to compute an output for the next node.

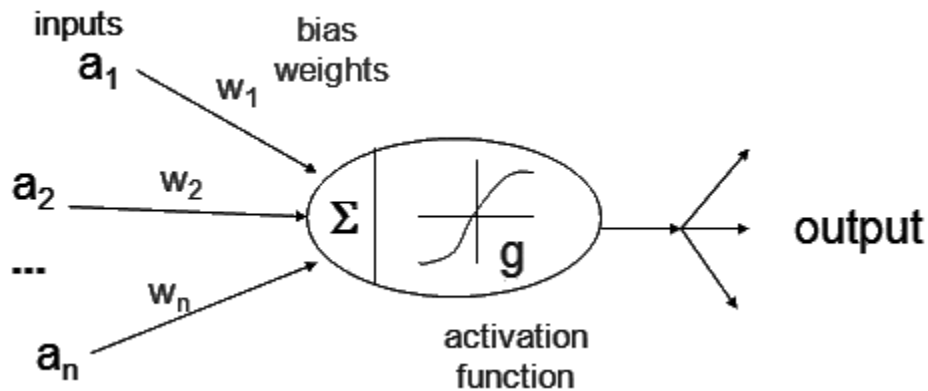


Figure 9. A Single Neuron #2

A **weight** (**w**) is associated to it, which takes value depending on the relevance to the other inputs. The function **f** for the node is as above. There is also another input called bias **b**, associated with it. Its function is to provide every node with a trainable constant value.

The input of a single node can be expressed as the linear combination of the predecessor node values with respect to their weights and the bias value.

2.3.2 Activation Functions

The function f as you can see from the equation, is non-linear and is called an **Activation Function**. Its purpose is to compute the output of a node given an input or set of inputs.

The Activation Function takes a single number and applies some mathematical operations on it. There are many of them, but here are some of the most popular Activation Functions:

2.3.2.1 Sigmoid Activation Function

A sigmoid function also called a logistic or logit function, squashes the y values in a range between 0 and 1, so they can be used in expressing probabilities. They are used to produce the output of a neuron. Their domain is in \mathbb{R}^* and their response value is commonly monotonically increasing or decreasing. Another commonly used range is between -1 and 1.

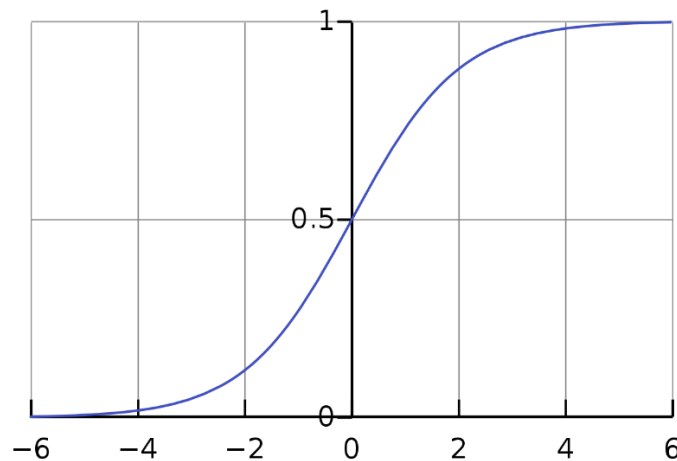


Figure 10. Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

2.3.2.2 Tanh Activation Function

The \tanh function (the hyperbolic tangent function) is a rescaled sigmoid function. This means it takes a real-valued input and squashes it to the range $[-1, 1]$.

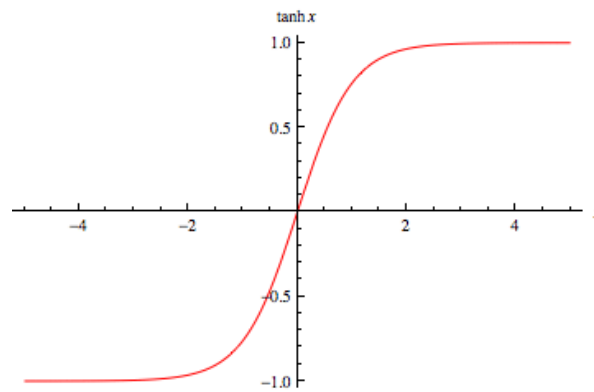


Figure 11. Tanh function

$$\tanh(x)$$

2.3.2.3 ReLU Activation Function

ReLU outputs zero for all $x \leq 0$, otherwise outputs the given value as it is.

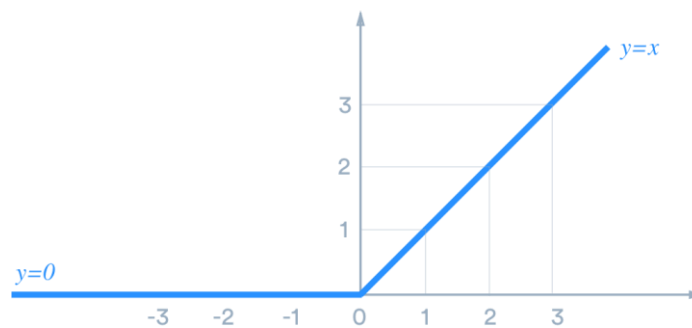


Figure 12. ReLU

$$\max(0, x)$$

2.3.3 Regression

2.3.3.1 Linear Regression

Linear Regression (LR) is one type of approach to modeling relationship between a scalar response and one or more descriptive variables. It may be a **simple linear regression** or **multiple linear regression**.

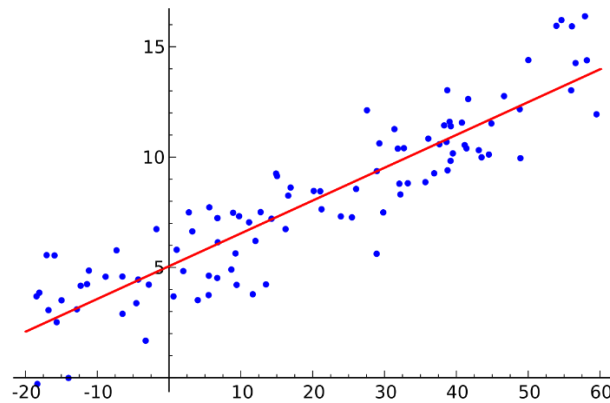


Figure 13. Linear Regression #1

Real data is a lot more complicated than this, so this simple model won't be able to find the complex patterns in the real data. To solve this problem, we use non-linear (curved) functions.

To have our model to be training more and more with our data, we add more hidden layers to our network. But there's a problem that comes with this, called the **Vanishing Gradient Problem**. **Gradient** is a numeric calculation that allows us to know how to adjust the parameters of the network in such a way that its output deviation is minimized. The Vanishing Gradient Problem occurs when during the back-propagation step the gradients become smaller and smaller, until eventually they vanish. No gradients, means no learning. This happens because of these sigmoid functions squeezing information.

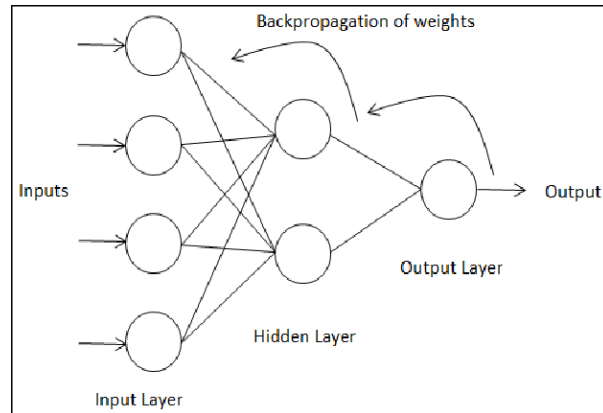


Figure 14. Back Propagation

Decay of information through time

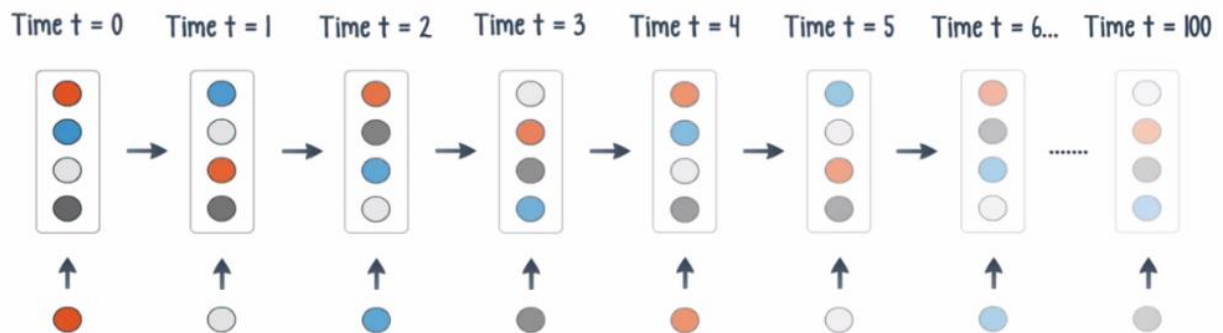


Figure 15. Vanishing Gradient Problem

To understand the rate of change of sigmoid function with respect to gradient value changes, we must take the derivative of the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)[1 - \sigma(x)]$$

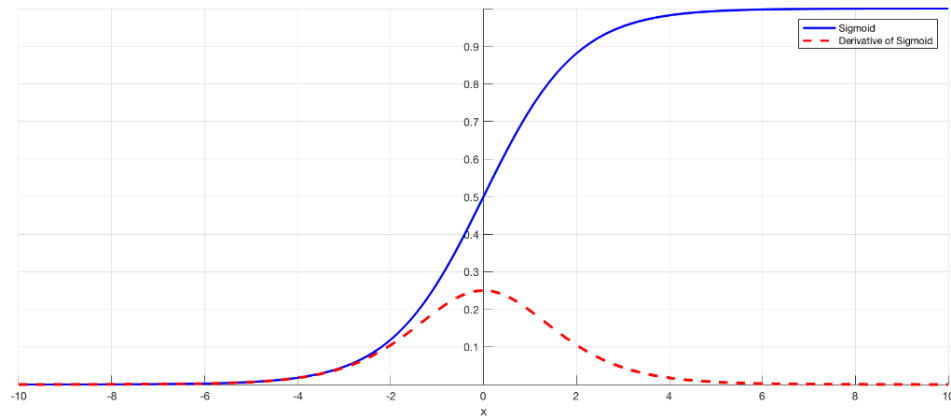


Figure 16. Sigmoid Derivative

We can now see the nature of change of the gradient. At points with large positive or negative value, the gradient comes really close to 0.

This isn't good since in our neural networks the gradients are back-propagated, which means these gradients are used again in the previous neurons from which they came from. When the gradients become 0, there is no learning, hence they are useless. This is the squeezing nature of sigmoid functions and the same happens even when using `arctan()` functions which squeezes real numbers between -1 and 1.

This problem can be solved by using ReLU. But in a random case we may face another problem which is similar to the Vanishing Gradient Problem. This time we have the Dying ReLU problem which returns 0 for $x \leq 0$.

If $W_x + b$ is the input for one neuron, then in ReLU we have:

$\text{ReLU}(W_x + b) = 0$, if $\max(W_x + b, 0) = 0$, i.e. $W_x + b < 0$, i.e. $b < -W_x$.

This means that if our inputs are not big enough to overcome the bias, these neurons remain dead. Dead neurons don't learn.

This means we should work with the left part of the ReLU function. We should include a negative case so our inputs are not blocked during the back-propagation step.

We can slightly leak information on the left part of the ReLU where the output is always 0.

We can make learning possible to these dead neurons by using Leaky ReLU or ELU activation functions.

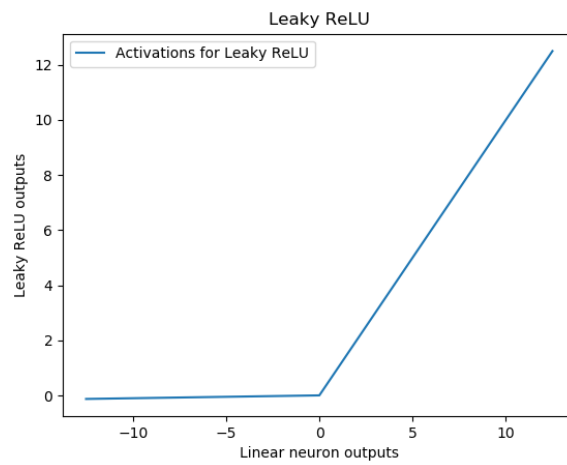


Figure 17. Leaky ReLU

$$f(x) = \begin{cases} 0,01x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$$

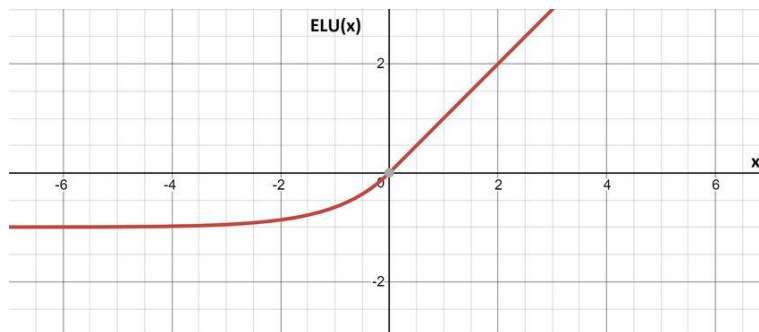


Figure 18. ELU

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

But, what activation function should we use on the output neurons? That depends on the problem we're solving.

Our second model will be built using Softmax Regression. The number of neurons in the output layer is equal to the number of classes we are dealing with, and the values are the probabilities of belonging to a particular class.

2.3.3.2 Softmax Regression

Given vector $\vec{z} = [z_1, \dots, z_n]$, the **Softmax** function squashes \vec{z} values between the range $[0, 1]$ and all the resulting elements add up to 1. It is applied to the output score s . These elements represent class probabilities.

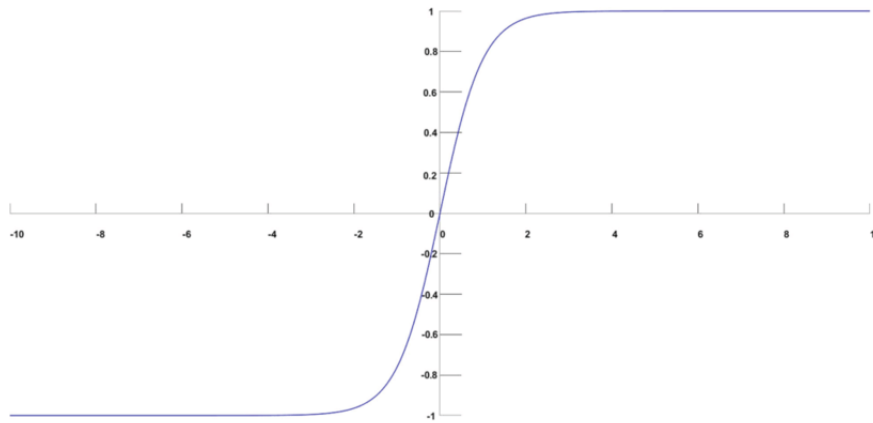


Figure 19. Softmax function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Let's take a look at this simple case of a neural network with one hidden layer and two outputs:

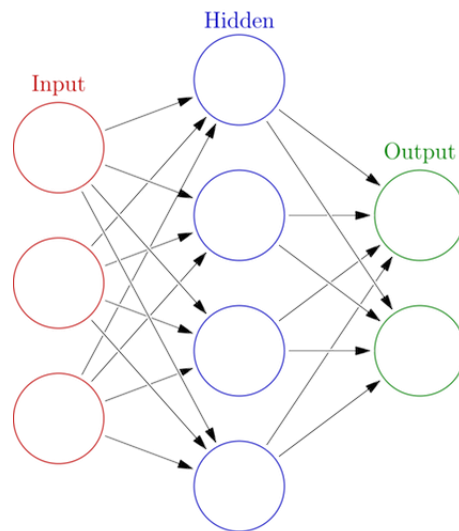


Figure 20. Simple Neural Network

The hidden layer neurons have no activation functions at all and the output neurons are Softmax activated.

This leads to a line separator.

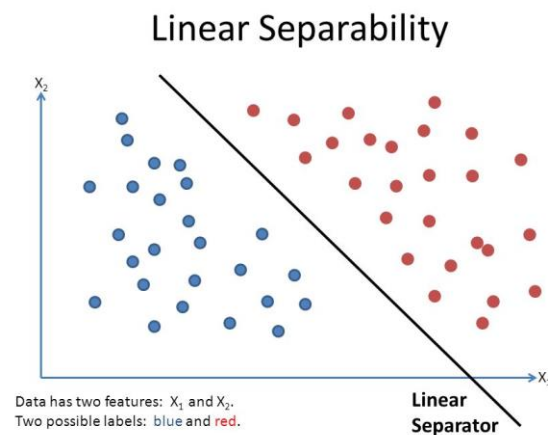


Figure 21. Linear Separability

Why is this boundary a line? To understand that, let o be the output.

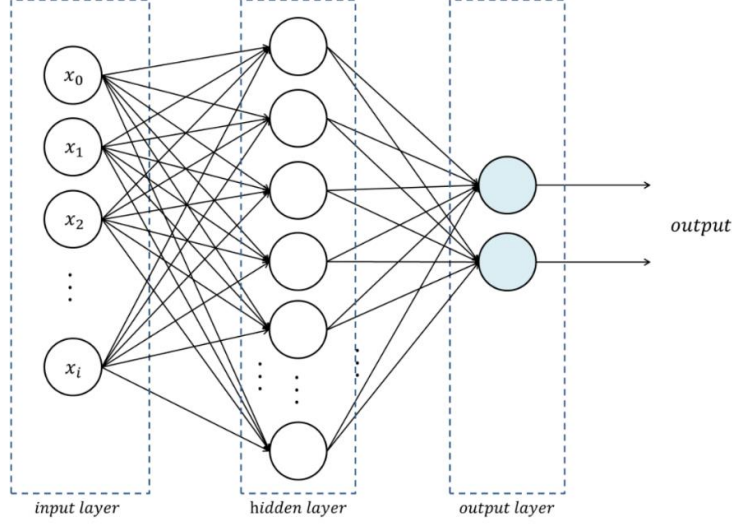


Figure 22. Multi-layered Neural Network

Then, $o = W_2h + b_2$, where W_i is the sum of all weights in the i -th layer, b_i is the bias in the i -th layer, and h is the output of the hidden layer.

But we know that we have $h = W_1x + b_1$, where x is the linear combination of the input values, so we substitute that in the equation above and we get the following:

$$o = W_2(W_1x + b_1) + b_2$$

$$o = W_2W_1x + W_2b_1 + b_2$$

Let us define $U = W_1W_2$ and $V = W_2b_1 + b_2$. Then, we get

$$o = Ux + V$$

This is a linear equation. Now this output is passed to a softmax activation. Since there's two neurons, which we could call them o_1 and o_2 , the output activation of the first could be written as:

$$\text{softmax}(o_1) = \frac{e^{o_1}}{e^{o_1} + e^{o_2}}$$

We divide both the numerator and denominator by e^{o_1} and we get:

$$\text{softmax}(o_1) = \frac{1}{1 + e^{o_2 - o_1}}$$

We substitute the value of the output equations to o_1 and o_2 :

$$\text{softmax}(o_1) = \frac{1}{1 + e^{(U_1x + V_1) - (U_2x + V_2)}} = \frac{1}{1 + e^{(U_1 - U_2)x - (V_1 - V_2)}} = \frac{1}{1 + e^{(U_0x + V_0)}}$$

$U_1 - U_2$ and $V_1 - V_2$ can be written down as just some constants U_0 and V_0 .

It looks clearly that this essentially boils down to a sigmoid function.

2.3.4 Loss Functions

Now let's explain the Loss or Cost functions used in this model.

In the optimization algorithm context, the function used to evaluate an alternative solution is known as the objective function.

We may want to minimize or maximize this objective function, meaning we are seeking another solution that has the highest or the lowest score respectively.

In our case we want to minimize the error. That's why we refer to as the Cost function or **Loss Function** and the value calculated by it is referred to as the loss.

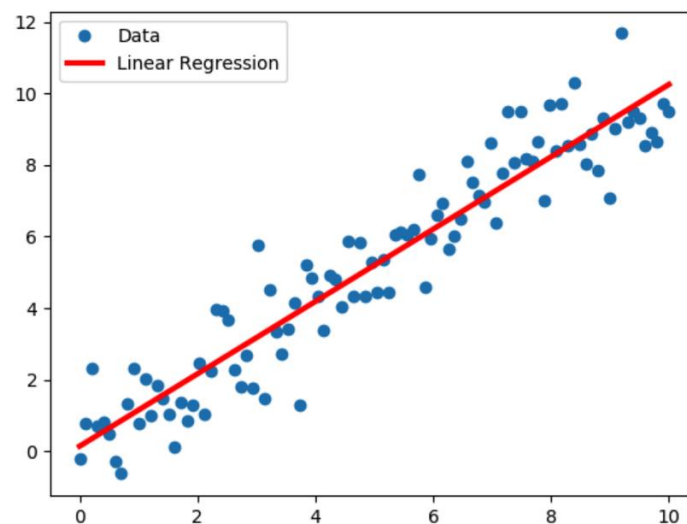


Figure 23. Linear Regression #2

In Regression models we build the scatterplot to simply generalize from the data plotted in the graph. But we want to calculate the general loss value of this regression line. We need to do this because we want to test our model how well it can predict (generalize) an output from a randomly given input.

The scatterplot is built by minimizing the general error distances.

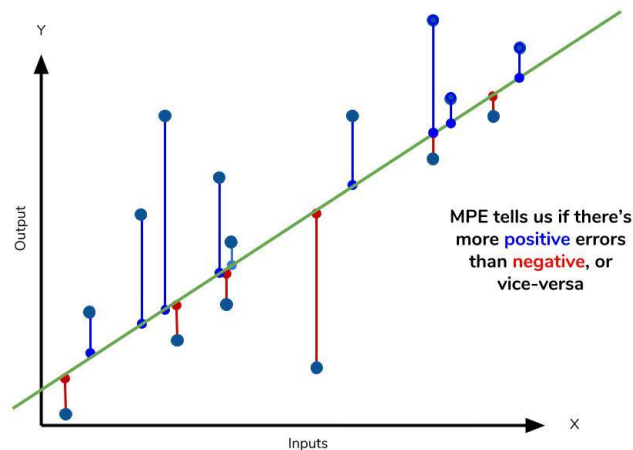


Figure 24. Mean Percentage Error

Let's first calculate the loss for one prediction only. Let's denote the real value of the output with y and the predicted value with \hat{y} .

Then the distance between these two would sort of give us an accuracy how well did our model perform. We would formulate the error for a case like below:

$$\text{error} = y - \hat{y}$$

2.3.4.1 MSE: Mean Squared Error

But what if we are dealing with negative numbers? In that case, the errors of equally distant data points from the line on opposite sides of it would cancel out and we would mistakenly be led to believe that the model is incurring no loss, but in fact, that is not the case. In order to handle this issue, we use the mean squared error which is defined as:

$$\text{error} = (y - \hat{y})^2$$

But this approach has its own downfalls, like for example if we have outliers in the graph like shown below. If these outliers are far away from the regression line, meaning they are large values, then squaring these really large values could lead to our regression line to shift up and lack generalization. They could be a distraction, or maybe they may have a significance for the model.

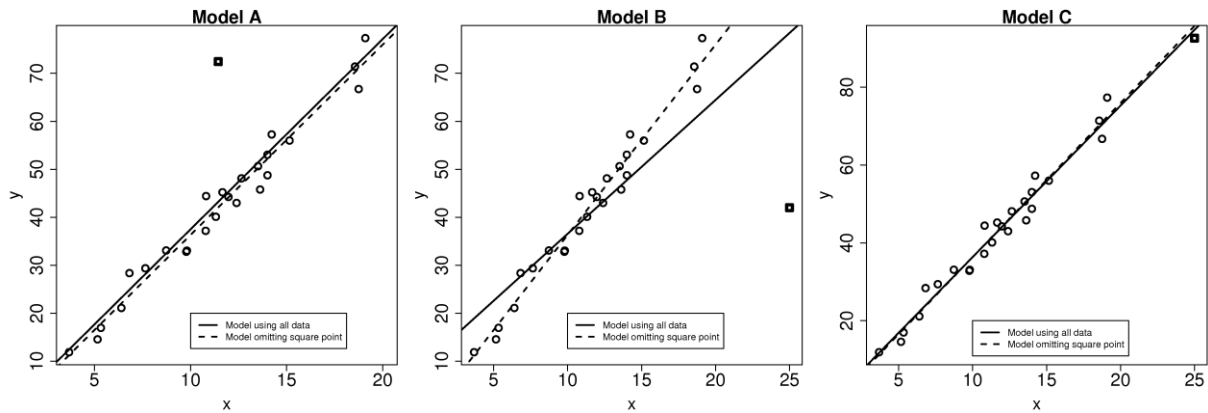


Figure 25. Outliers Effect

2.3.4.2 MAE: Mean Absolute Error

We can solve this by taking the absolute distance between y and \hat{y} . That is:

$$\text{error} = |y - \hat{y}|$$

$$MAE = \frac{1}{n} \sum \underbrace{|y - \hat{y}|}_{\text{The absolute value of the residual}}$$

Divide by the total number of data points (points to $\frac{1}{n}$)
Actual output value (points to y)
Predicted output value (points to \hat{y})
Sum of (points to \sum)
The absolute value of the residual (points to $|y - \hat{y}|$)

Figure 26. Mean Absolute Error

Also, this approach has its downfalls. What if these outliers are crucial to the model? Then we are ignoring them and therefore this leads to poor prediction or poor generalization. These two issues are widely experienced while designing machine learning models. They are what are called overfitting and underfitting. In any case, MSE or MAE are usually the way to go with regression. Support Vector Regression uses this type of loss function.

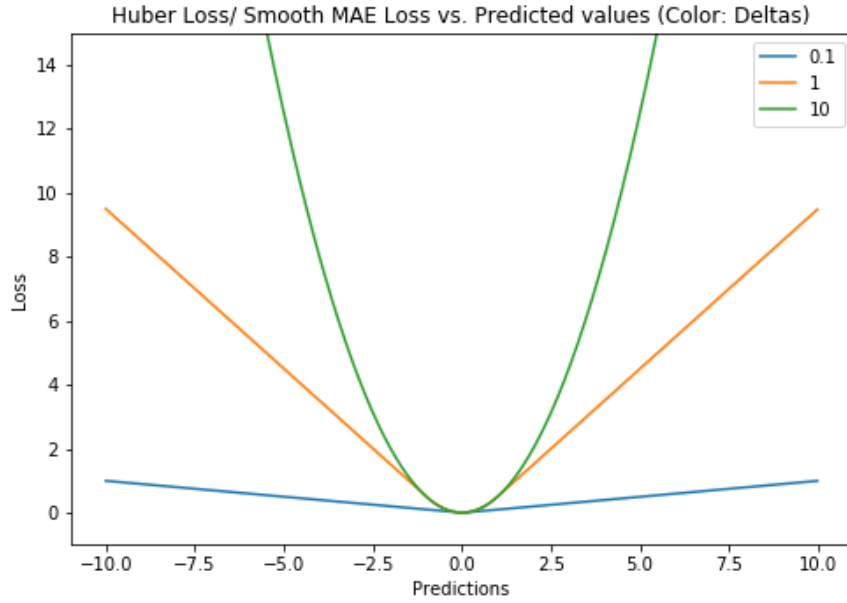


Figure 27. Huber Loss vs. Smooth MAE and Predicted Values

2.3.4.3 Pseudo - Huber Loss

Another Loss function besides Squared and Absolute Distance Loss, is the Pseudo-Huber Loss.

This type of loss function takes the best of both losses. That means it is a piecewise function like shown below:

$$L_{\delta}[y, f(x)] = \begin{cases} \frac{1}{2}[y - f(x)]^2, & \text{for } |y - f(x)| \leq \delta, \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & \text{otherwise.} \end{cases}$$

2.3.4.4 Logarithmic Loss

In our case we are using the **Logarithmic Loss Function**, also known as Logistic Loss or Cross-Entropy Loss function. This is used in multinomial logistic regression and extensions of it such as neural networks. This function returns predicted \hat{y} probabilities for its training data y .

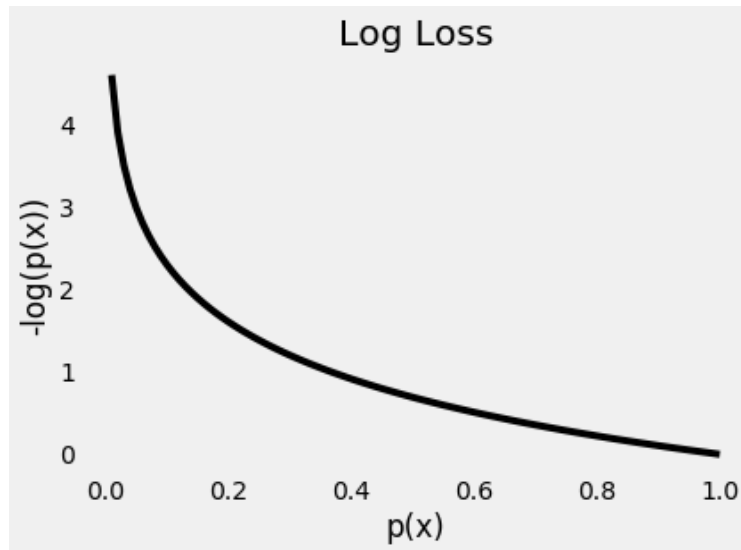


Figure 28. Logarithmic Loss

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log[p(y_i)] + (1 - y_i) \cdot \log[1 - p(y_i)]$$

Chapter 3

3.1 Overview

With all of these concepts in the math background explained, we are good to go with developing our deep learning model.

As I mentioned I will be using Keras as a framework to do this.

To do this I first open **Anaconda Navigator**, which is the most popular Data Science platform.

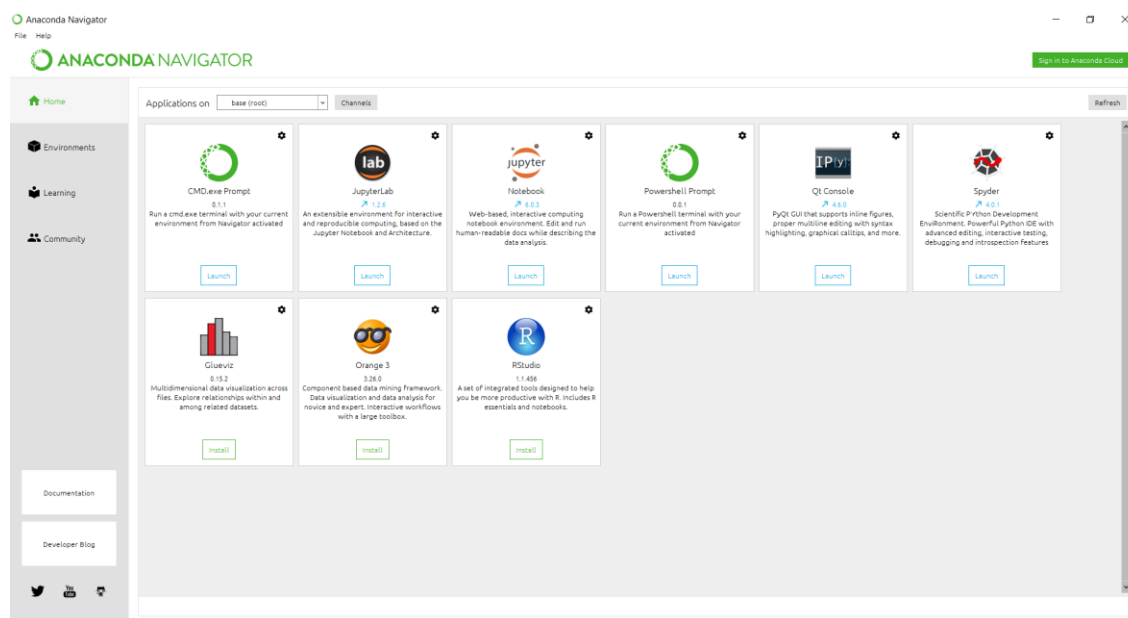


Figure 29. Anaconda Navigator

This platform offers many tools to help Data Scientists run their experiments and analyze data.

I will be using Jupyter Notebook as I mentioned, because it offers me a really responsive and interactive development environment and I can execute my code step by step to see the results and if it has any errors.



Figure 30. Jupyter Notebook Environment

Then I navigate to my project directory, in this case is: **Desktop/Carcinoma**.

Inside this folder there are 3 files, the Jupyter Notebook file (**.ipynb**), and the two dataframes (**.csv**), which make up the whole dataset.



Figure 31. Project Directory on Jupyter Notebook Environment

3.1.1 matrix_of_features_x.csv file:

This file contains all the input features I will be using to train and test the model.



Figure 32. Matrix of Features X

3.1.2 matrix_of_labels_y.csv file:

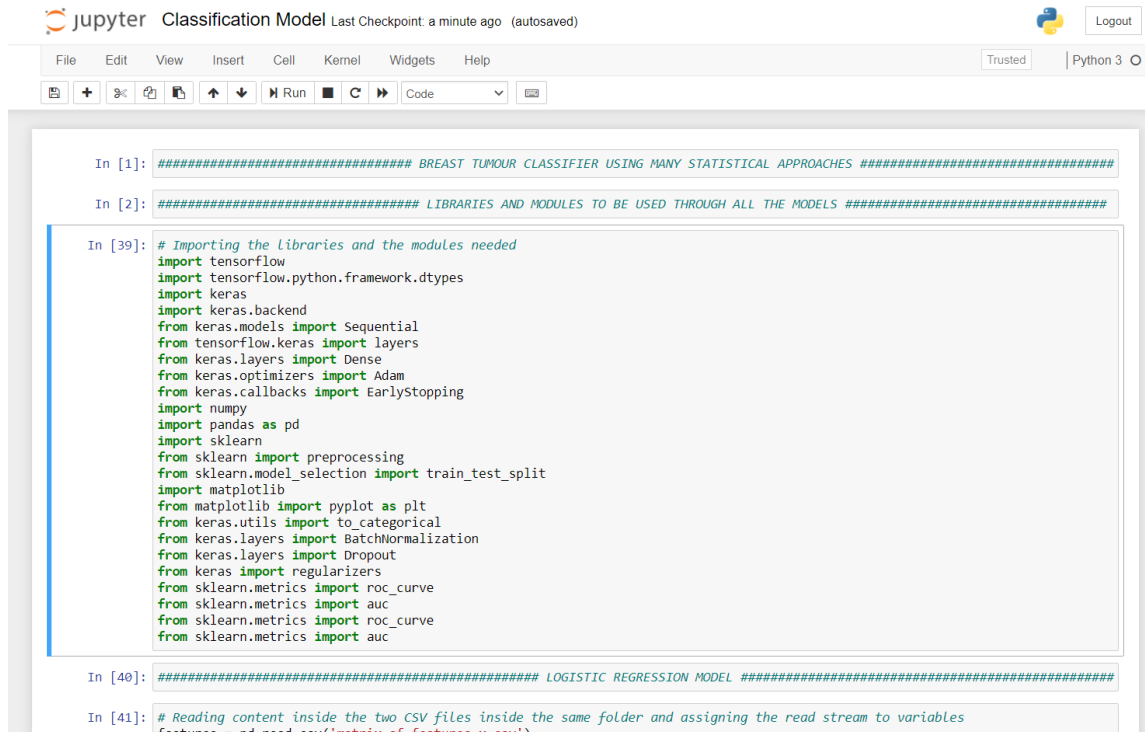
This file contains all the labels, or classification digits that the model will use to train and test itself.



Figure 33. Matrix of Labels Y

3.1.3 The Classification Model.ipynb file:

This file is the software I developed. It is executable only from the Jupyter Notebook interpreter.



The screenshot shows a Jupyter Notebook interface with the title 'Classification Model' and a status bar indicating 'Last Checkpoint: a minute ago (autosaved)'. The notebook contains several code cells. The first cell is a comment: '##### BREAST TUMOUR CLASSIFIER USING MANY STATISTICAL APPROACHES #####'. The second cell is another comment: '##### LIBRARIES AND MODULES TO BE USED THROUGH ALL THE MODELS #####'. The third cell, labeled 'In [39]:', contains a block of Python code that imports various libraries and modules. The code is as follows:

```
# Importing the libraries and the modules needed
import tensorflow
import tensorflow.python.framework.dtypes
import keras
import keras.backend
from keras.models import Sequential
from tensorflow.keras import layers
from keras.layers import Dense
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
import numpy
import pandas as pd
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import matplotlib
from matplotlib import pyplot as plt
from keras.utils import to_categorical
from keras.layers import BatchNormalization
from keras.layers import Dropout
from keras import regularizers
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

The fourth cell, labeled 'In [40]:', contains a comment: '##### LOGISTIC REGRESSION MODEL #####'. The fifth cell, labeled 'In [41]:', contains a comment: '# Reading content inside the two CSV files inside the same folder and assigning the read stream to variables' followed by a line of code: `features = pd.read_csv('matrix of features v.csv')`.

Figure 34. Classification Model File

3.2 Steps followed to create the models:

Before explaining how the code works, I want to make known a list of steps I followed to create this model:

1. First, I import some libraries I will be needing throughout the implementation of the network.
2. Then I import the dataframes.
3. I apply Standardization Scaling on the values of the features.
4. I then split the dataframes into a total of 4 parts.
5. I then construct the Neural Network.
6. I define an Early Stopper.
7. I calculate the Loss and Accuracy of the Training Set and Validation Set.
8. I calculate the Loss and Accuracy of the Testing Set.
9. I calculate the AUC Score of Testing Set.
10. I plot the ROC Curve of the Testing Set.
11. I calculate the AUC Score of the Training Set.
12. I plot the ROC Curve of the Training Set.

3.3 Model #1: Logistic Regression Model

3.3.1 Importing the Libraries

Since we are working with Keras, we need to import Tensorflow first, then the Keras itself.

```
In [39]: # Importing the libraries and the modules needed
import tensorflow
import tensorflow.python.framework.dtypes
import keras
import keras.backend
from keras.models import Sequential
from tensorflow.keras import layers
from keras.layers import Dense
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
import numpy
import pandas as pd
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import matplotlib
from matplotlib import pyplot as plt
from keras.utils import to_categorical
from keras.layers import BatchNormalization
from keras.layers import Dropout
from keras import regularizers
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

Figure 35. Importing the Libraries

We will also be working with vectors and matrices, so we will be needing Numpy.

To work with dataframes, reading and splitting, we will need Pandas.

Most importantly we will be classifying, so we import Scikit-Learn also.

Last but not least, we import Matplotlib to graph our curves.

3.3.2 Importing the Dataframes

We import the dataframes using Pandas's method called `read_csv ()` and we assign them to two variables: features and labels.

```
In [62]: ##### LOGISTIC REGRESSION MODEL #####

In [63]: # Reading content inside the two CSV files inside the same folder and assigning the read stream to variables
features = pd.read_csv('matrix_of_features_x.csv')
labels = pd.read_csv('matrix_of_labels_y.csv')
```

Figure 36. Importing the Dataframes

3.3.3 Standardization Scaling the Features

Since we will work with probabilities, we need to work with numbers who lay in a range between 0 and 1. That's why we standardize them by using the scale method from the preprocessing class.

```
In [64]: # Standardizing the data points, by putting them on a scale
features = preprocessing.scale(features)
```

Figure 37. Standardization Scaling the Features

3.3.4 Splitting of the Dataset

To be able to use a part of our data for training we should perform the splitting process on both our dataframes.

We split the matrix of features in 80% of it being used for training and 20% of it being used for testing. We do the same for the matrix of labels. Each row on the labels dataframe corresponds to the respective row on the matrix of features. So, our model will use 80% of the data to be trained and 20% to be tested how well did the model train.

```
In [65]: # Splitting both datasets into training and testing dataframes with testing data size 20% and the rest being used for training
xtr, xts, ytr, yts = train_test_split(features, labels, test_size = 0.2)
```

Figure 38. Splitting the Dataset

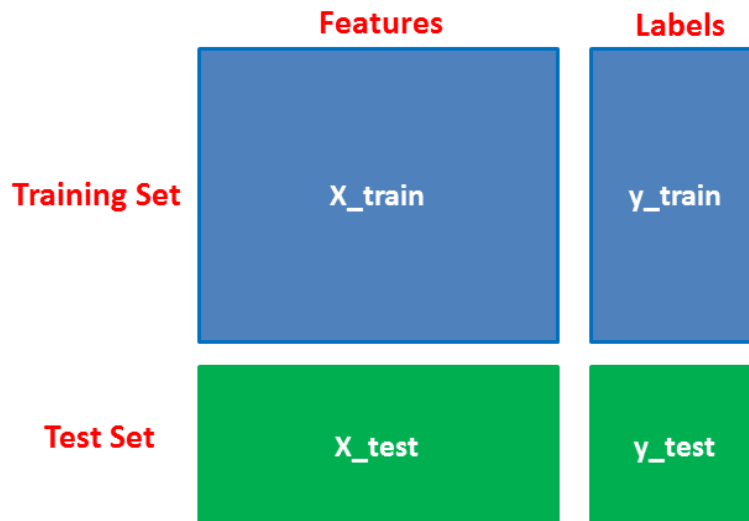


Figure 39. Dataset Splitting Schema

3.3.5 Constructing the Neural Network

Then, we instantiate an object of the `Sequential ()` class. This will allow us to use the method `add (Dense ())`, which will construct a neural network with as many hidden layers as we want.

```
In [66]: # Constructing the Logistic Regression Model using a Neural Network
model = Sequential()
model.add(Dense(21, input_shape = (30, ), activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = Adam(lr = 0.001), metrics = ['accuracy'])
```

Figure 40. M#1: Constructing the Neural Network

In our case we have a single hidden layer with 21 neurons, an input layer with 30 neurons (number of features), and we have chosen to use the ReLU activation function.

We then define the output layer with a single neuron and the activation function of sigmoid. This output neuron will take as an input from the hidden layer the probability of being a particular class. This number will then be squashed in the sigmoid function so that we can determine if it is closer to 0 (benign tumor), or 1 (malignant tumor).

We should also call the `compile ()` method which is responsible for operations during compilation time, such as loss functions, optimizers and metrics.

We have chosen as we explained before, the **Binary Cross-Entropy** method of calculating our loss. We use an **Adam optimizer**, which is an optimizing algorithm that can be used to tune the network's weights based in training data.

We then pass the last parameter which has to do with metrics. We want the loss and accuracy to be stored somewhere as variables. That's what `metrics = ['accuracy']` parameter does, it logs the accuracy value in a History object which we will use below.

3.3.6 Early Stopper

Another important step in training our model is to define an **Early Stopper**. This will make possible the halting of our training process at the right time, so that the model does not overfit or underfit.

```
In [67]: # Defining an Early Stopper that will train our model in 2000 epochs
estop = EarlyStopping(monitor = 'val_loss', min_delta = 0, patience = 15, verbose = 1, mode = 'min')
fitted_model = model.fit(xtr, ytr, epochs = 2000, validation_split = 0.15, verbose = 0, callbacks = [estop])
history = fitted_model.history
print(fitted_model.history.keys())

Epoch 00093: early stopping
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

Figure 41. M#1: Early Stopper

Early Stopper allows you to specify a randomly large number of training epochs and stop the training process once the model performance stops improving. The performance can be checked using a validation set. The validation set is a subdivision of the training set used to tune the parameters of a classifier, like choosing the number of hidden layers or neurons in the network, etc.

In our case the validation split is **0.15**, or **15%** of the training set.

We are using 2000 epochs for this training. This means that the cycles or iterations the model will perform on this dataset will be 2000 times. In other words, the dataset will iterate 569 rows 2000 times.

We define a history variable and we assign that the history or metrics we retrieve from our fitted or already trained model, in order to understand the model performance.

3.3.7 Loss and Accuracy of the Training Set and Validation Set

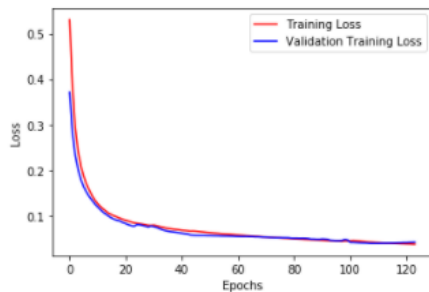
This history variable contains an array of variables such as `val_loss` and `val_accuracy`, which is the loss and accuracy during validation process; `loss` and `accuracy`, which are the loss and accuracy during training process.

```
In [46]: # Plotting the Loss of Training and Validation dataframes over the epochs
loss = history['loss']
plt.figure()
val_loss = history['val_loss']
plt.figure()
plt.plot(loss, 'r', label = 'Training Loss')
plt.plot(val_loss, 'b', label = 'Validation Training Loss')
plt.legend()
plt.ylabel("Loss")
plt.xlabel("Epochs")

# Plotting the accuracy of Training and Validation dataframes over the epochs
acc = history['accuracy']
plt.figure()
val_acc = history['val_accuracy']
plt.figure()
plt.plot(acc, 'r', label = 'Training Accuracy')
plt.plot(val_acc, 'b', label = 'Validation Training Accuracy')
plt.legend()
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
```

Out[46]: Text(0.5, 0, 'Epochs')

<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

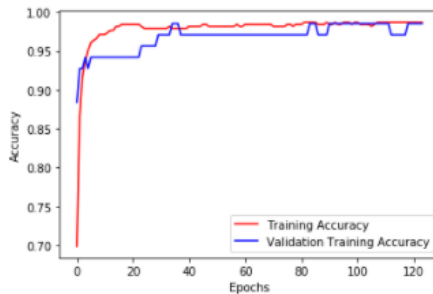


Figure 42. M#1: Loss and Accuracy of the Training Set and Validation Set

We then assign these history variable values to some other variables, for brevity, and we plot their graphs by noting them with different colors. We group losses and accuracies together.

As we can see the validation loss and the training loss is exponentially decreasing. This is a good sign. This means that the values that our model can predict are not far enough from the real values, also called the **ground truth**.

On the other hand, we have the accuracies of both training and validation. The accuracy in our case is the number of correct diagnoses out of all total diagnoses. We can see that they are increasing logarithmically. This means that the accuracy of training the data is getting higher and has stopped at the right time.

3.3.8 Loss and Accuracy of the Testing Set

We then calculate the loss and accuracy of the data tested, and we see that the loss is really small and accuracy really high. This is a great performance and the model is generalizing pretty well.

```
In [69]: # Calculating the loss and accuracy of data tested
loss, acc = model.evaluate(xts, yts)
print("Testing Data Loss: ", loss)
print("Testing Data Accuracy: ", acc)

114/114 [=====] - 0s 43us/step
Testing Data Loss: 0.04298649496284493
Testing Data Accuracy: 0.9912280440330505
```

Figure 43. M#1: Loss and Accuracy of the Testing Set

Next, we will use two other metrics to further reveal the performance of the model. Before explaining those, I want to explain some other concepts which are in the way.

Depending on our intentions, we want to maximize certain parameters and minimize others. We may formulate two cases of minimization or maximization here:

		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	TP True Positive	FP False Positive
	negatives	FN False Negative	TN True Negative

Figure 44. Cases Schema

TP: True Positive | Correctly predict something to be true.
(*Tumor is malignant and we predict to be malignant*).

TN: True Negative | Correctly predict that something is false.
(*Tumor is benign and we predict to be benign*).

FP: False Positive | Incorrectly predict that something is true.
(*Tumor is benign, and we predict to be malignant*).

FN: False Negative | Incorrectly predict that something is false.
(*Tumor is malignant, but we say is benign*).

Case 1: Maybe we would only want to make sure that a patient has cancer, without further diagnosis. To do so we could maximize True Positives and minimize False Negatives, even if it causes an increase in False Positives and decrease in True Negatives.

Case 2: On the other hand, maybe we would want to raise the diagnosis threshold, in order to collect less patients to be tested, and collect only those with IDC. This means we could maximize True Negatives and Minimize False Positives, even if it causes increase in False Negatives and decrease in True Positives.

We want to see the both scenarios how they perform in a single graph. That is exactly what a ROC Curve does.

AUC is the Area Under the Curve. It is a numerical value that calculates how the model performs in these situations where we raise or lower the threshold to maximize or minimize the chances of a patient being most likely diagnosed with IDC or not.

The ROC Curve plots all these threshold values of the trained model.

On the **y-axis** we have the TPR (True Positive Rate) and on the **x-axis** we have the FPR (False Positive Rate).

$$\text{TPR} = (\text{no. of correctly predicted positives}) / (\text{all actual positives}),$$

$$\text{FPR} = (\text{no. of incorrectly predicted positives}) / (\text{all actual negatives}).$$

Let's say we choose our threshold to be **0.3**.

That means any patient with the probability below **0.3**, the model would predict that the patient is most likely to have a benign tumor.

And the patient with the probability above **0.3** has a malignant tumor.

This model has more **sensitivity**, because if the model's diagnosis predicts that the tumor is benign, then the correct diagnosis has higher chances of being benign also. This scenario would best fit **Case 1** we covered before, where we wanted to make sure if the patient had IDC without requiring further diagnosis.

Then we will tune the threshold by raising it to **0.7** and calculate the True and False Positive Rates. This is a **Specific** Model. This means that if the diagnosis by the model is positive, then it is most likely that the correct diagnosis is also positive. This case would suit more **Case 2** scenario from the scenarios we covered previously.

3.3.9 AUC Score of Testing Set

We then plot the points obtained by calculating the Rates for each prediction and calculate the area under the curve.

This is done through the **roc_curve ()** method.

```
In [70]: # Calculating the AUC score of Testing data
yts_pred = model.predict_proba(xts)
fal_pos_rate, tru_pos_rate, thresh = roc_curve(yts, yts_pred)
auc_krs = auc(fal_pos_rate, tru_pos_rate)
print('Testing Data AUC: ', auc_krs)

Testing Data AUC:  0.9992766726943942
```

Figure 45 M#1: AUC Score of Testing Set

It measures the performance across all possible classification thresholds. In our case the score is **0.99**, which is close to the ideal AUC score of 1.

3.3.10 ROC Curve of the Testing Set

This is the ROC Curve we obtained by plotting all the True and False Positive Rates we get for each prediction, by tuning the threshold from a sensitive to a specific one.

```
In [71]: # Plotting the ROC curve of Testing data
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fal_pos_rate, tru_pos_rate, label = 'Keras (area = {:.3f})'.format(auc_krs))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc = 'best')
plt.show()
```

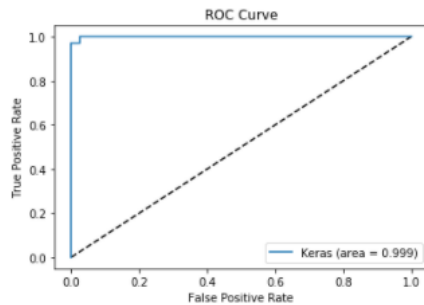


Figure 46. M#1: ROC Curve of the Testing Set

Usually in medical diagnoses, test **sensitivity** is the ability of a test to correctly identify those with the disease (TPR), whereas test **specificity** is the ability of the test to correctly identify those without the disease (FPR).

3.3.11 AUC Score of the Training Set

Then we repeat this process of calculating AUC Score for the training set also.

```
In [72]: # Calculating the AUC score of Training data
ytr_pred = model.predict_proba(xtr)
fal_pos_rate, tru_pos_rate, thresh = roc_curve(ytr, ytr_pred)
auc_krs = auc(fal_pos_rate, tru_pos_rate)
print('Training Data AUC: ', auc_krs)
```

Training Data AUC: 0.9991415958142577

Figure 47. M#1: AUC Score of the Training Set

3.3.12 ROC Curve of the Training Set

And we also repeat the process of plotting ROC Curve for the training set.

```
In [73]: # Plotting the ROC curve of Training data
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fal_pos_rate, tru_pos_rate, label='Keras (area = {:.3f})'.format(auc_krs))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc = 'best')
plt.show()
```

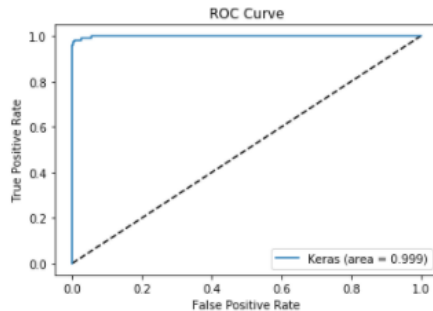


Figure 48 M#1: ROC Curve of the Training Set

3.3.13 Conclusion

In this model we used Sigmoid as an activation function and Binary Cross-Entropy as a Loss Function.

This model would be considered a **Logistic Regression Model** and it is just one type of approach to a binary classification problem like this.

3.4 Model #2: Softmax Regression Model

Now let's observe how our model would behave with **Softmax** as an Activation Function, and **Categorical Cross-Entropy** as a Loss Function.

This model is called a **Softmax Regression Model** and can be used in multiclass classification problems, while Logistic Regression can't.

3.4.1 Converting Integer Array to Binary Class Matrix

We will first need to convert our training set label integers from 0 and 1 to a binary class matrix. We will do this by using the Numpy utility called `to_categorical()`.

```
In [74]: ##### SOFTMAX REGRESSION MODEL #####

In [75]: # Converting Matrix of Labels Y into categorical type of data
ytr_categ = to_categorical(ytr)

In [76]: print(ytr)

      1
523  0
179  1
115  0
 57  0
369  1
.. ..
 83  0
 76  1
431  1
 84  1
 27  1

[454 rows x 1 columns]
```

Figure 49. M#2: Converting Integer Array to Binary Class Matrix #1

CLASS 1 = 0

CLASS 2 = 1

`[1, 0]` represents **CLASS 1**, which is **0**.

`[0, 1]` represents **CLASS 2**, which is **1**.

Figure 50. M#2: Converting Integer Array to Binary Class Matrix #2

To understand a bit better, let's take an example when we have three classes:

CLASS 3 = 2

0, 1, 2, 0

 $[1, 0, 0], \quad [0, 1, 0], \quad [0, 0, 1], \quad [1, 0, 0]$

Figure 51. Multi-Class vs. Multi-Label Schema

The reason we did this converting is that the activation function (softmax) and the loss function (categorical cross-entropy) we will use require data to be in this format.

3.4.2 Constructing the Neural Network

Now, we are good to go. We define the model like we did in the Logistic Regression example. Then we add an input layer with 30 neurons (input features), and a hidden layer with 21 neurons, and an activation function type of softmax.

Now this time, these output neurons from the hidden layer connect with not one like we saw before, but two output neurons in the output layer. The output layer neurons use softmax activation again. That is why we used `to_categorical()` method. To match the training data format to the model's output formats would be in.

We then define the loss and optimizer like we did in the first example.

```
In [78]: # Constructing the Softmax Regression Model using a Neural Network
model = Sequential()
model.add(Dense(21, input_shape = (30, ), activation = 'softmax'))
model.add(Dense(2, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = Adam(lr = 0.0001), metrics = ['accuracy'])
```

Figure 52. M#2: Constructing the Neural Network

3.4.3 Early Stopper

We then train the model with the Early Stopping callback. Assign it to `fitted_model` variable, and assign `fitted_model.history` to `history` variable.

```
In [79]: # Defining an Early Stopper that will train our model in 2000 epochs
estop = EarlyStopping(monitor = 'val_loss', min_delta = 0, patience = 15, verbose = 1, mode = 'min')
fitted_model = model.fit(xtr, ytr_categ, epochs = 2000, validation_split = 0.15, verbose = 0, callbacks = [estop])
history = fitted_model.history
print(fitted_model.history.keys())

Epoch 01178: early stopping
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

Figure 53. M#2: Early Stopper

3.4.4 Loss and Accuracy of the Training Set and Validation Set

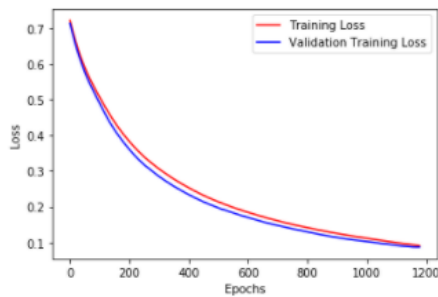
After that, I plot the training and validation loss and accuracy over the epochs altogether in one graph.

```
In [80]: # Plotting the loss of Training and Validation dataframes over the epochs
loss = history['loss']
plt.figure()
val_loss = history['val_loss']
plt.figure()
plt.plot(loss, 'r', label = 'Training Loss')
plt.plot(val_loss, 'b', label = 'Validation Training Loss')
plt.legend()
plt.ylabel("Loss")
plt.xlabel("Epochs")

# Plotting the accuracy of Training and Validation dataframes over the epochs
acc = history['accuracy']
plt.figure()
val_acc = history['val_accuracy']
plt.figure()
plt.plot(val_acc, 'r', label = 'val_acc')
plt.plot(acc, 'b', label = 'acc')
plt.legend()
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
```

Out[80]: Text(0.5, 0, 'Epochs')

<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

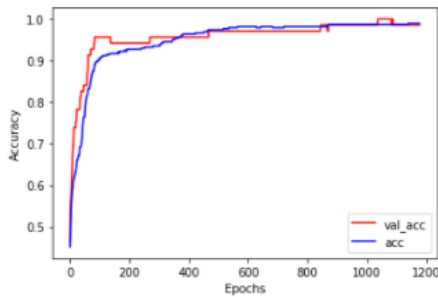


Figure 54. M#2: Loss and Accuracy of the Training Set and Validation Set

As we can see the losses and accuracies are exponentially decreasing and logarithmically increasing respectively.

3.4.5 Loss and Accuracy of the Testing Set

I also calculate the loss and accuracy of the data tested just like we did on the Logistic Regression Model, except this time I have to apply the `to_categorical()` method to the testing labels' data, because our model uses softmax regression as an activation function and the data should be of a categorical format.

```
In [81]: # Calculating the loss and accuracy of data tested
yts_cat = to_categorical(yts)
loss, acc = model.evaluate(xts, yts_cat)
print("Test Loss: ", loss)
print("Test Accuracy: ", acc)

114/114 [=====] - 0s 61us/step
Test Loss: 0.08059074102263701
Test Accuracy: 0.9912280440330505
```

Figure 55. M#2: Loss and Accuracy of the Testing Set

As we can clearly see the loss is pretty small and the accuracy is really high.

3.4.6 AUC Score and ROC Curve of the Testing Set

I still continue measuring the performance of the model with the AUC and the ROC Curve. I first define them for the data tested.

```
In [82]: # Calculating the AUC score of Testing data
yts_pred = model.predict_proba(xts)
fal_pos_rate, tru_pos_rate, thresh = roc_curve(yts, yts_pred[:,1])
auc_krs = auc(fal_pos_rate, tru_pos_rate)
print('Testing data AUC: ', auc_krs)

Testing data AUC: 0.9992766726943942
```

Figure 56. M#2: AUC Score of the Testing Set

```
In [83]: # Plotting the ROC curve of Testing data
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fal_pos_rate, tru_pos_rate, label = 'Keras (area = {:.3f})'.format(auc_krs))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc = 'best')
plt.show()
```

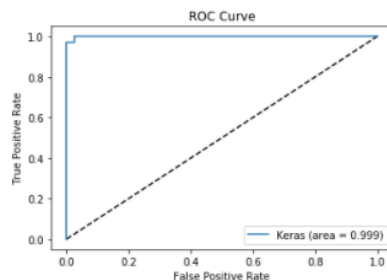


Figure 57. M#2: ROC Curve of the Testing Set

And once again, they are pretty good. When I calculate the AUC Score, I use the original y test and not the categorical y test, because this is the right format to go with.

3.4.7 AUC Score and ROC Curve of the Training Set

I do exactly the same thing for the training data.

```
In [84]: # Calculating the AUC score of Training data
ytr_pred = model.predict_proba(xtr)
fal_pos_rate, tru_pos_rate, thresh = roc_curve(ytr, ytr_pred[:,1])
auc_krs = auc(fal_pos_rate, tru_pos_rate)
print('Testing data AUC: ', auc_krs)
```

Testing data AUC: 0.9980788096795291

Figure 58. M#2: AUC Score of the Training Set

```
In [85]: # Plotting the ROC curve of Training data
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fal_pos_rate, tru_pos_rate, label = 'Keras (area = {:.3f})'.format(auc_krs))
plt.title('ROC Curve')
plt.legend(loc = 'best')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

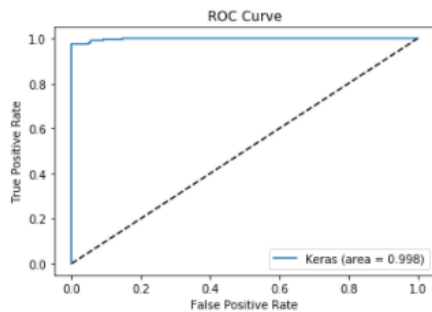


Figure 59. M32: Plotting the ROC Curve of the Training Set

Also, our training performance measurement is doing great.

3.5 Comparison: Logistic Regression vs. Softmax Regression

Let's now compare the Logistic Regression Model to the Softmax Regression Model to see if the performance has changed a bit.

1. In our Logistic Regression Model, the Loss and Accuracy of the data being tested was approximately 0.042986 and 0.99122, respectively. While, in our Softmax Regression Model, these were 0.08059 and 0.99122, respectively.
2. The AUC Scores of LR Model were 0.99927 for the testing and 0.99914 for the training. While, the AUC Scores of the Softmax Model were 0.99927 for the testing and 0.99807 for the training.

From the results we can judge that they have approximately the same performance in testing, training and validating the data.

We can now say that in this specific problem, the learning performance differences between the Logistic Regression or Softmax Regression Models is insignificant.

3.6 Model #3: Deep Learning Softmax Regression Model

I continue to experiment with the number of hidden layers in the model to see if it will improve the performance of the model. I again use Softmax Regression, but this time the learning type is called Deep Learning because we are dealing with many hidden layers.

3.6.1 Constructing the Neural Network

First, I define the model to be a `Sequential()` instance. I define the 30 inputs with 4 hidden layers containing each 21 neurons and 2 output neurons. Each of them having the same activation function. I use the same loss function and optimizer as I did before.

```
In [86]: ##### DEEP LEARNING SOFTMAX REGRESSION MODEL #####

In [87]: # Constructing a Deep Learning Softmax Regression Model using a Neural Network
model = Sequential()
model.add(Dense(21, input_shape = (30, ), activation = 'softmax'))
model.add(Dense(21, activation = 'softmax'))
model.add(Dense(21, activation = 'softmax'))
model.add(Dense(21, activation = 'softmax'))
model.add(Dense(2, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = Adam(lr = 0.001), metrics = ['accuracy'])
```

Figure 60. M#3: Constructing the Neural Network

3.6.2 Early Stopper

This time, since we have more hidden layers, let's also tune the epochs to be not 2000, but 3000 epochs.

```
In [88]: # Defining an Early Stopper that will train our model in 3000 epochs
estop = EarlyStopping(monitor = 'val_loss', min_delta = 0, patience = 20, verbose = 1, mode = 'min')
fitted_model = model.fit(xtr, ytr_categ, epochs = 3000, validation_split = 0.1, shuffle = True, verbose = 0, callbacks = [estop])
history = fitted_model.history
print(fitted_model.history.keys())

Epoch 00218: early stopping
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

Figure 61. M#3: Early Stopper

3.6.3 Loss and Accuracy of the Training Set and Validation Set

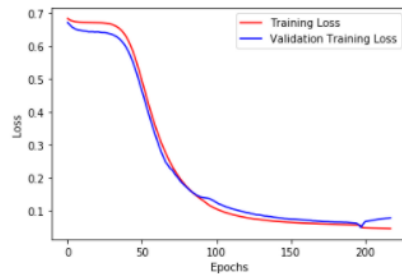
After that, we plot the losses and accuracies of the training and validation split, over the epochs.

```
In [89]: # Plotting the Loss of Training and Validation dataframes over the epochs
loss = history['loss']
plt.figure()
val_loss = history['val_loss']
plt.figure()
plt.plot(loss, 'r', label = 'Training Loss')
plt.plot(val_loss, 'b', label = 'Validation Training Loss')
plt.legend()
plt.ylabel("Loss")
plt.xlabel("Epochs")

# Plotting the accuracy of Training and Validation dataframes over the epochs
acc = history['accuracy']
plt.figure()
val_acc = history['val_accuracy']
plt.figure()
plt.plot(val_acc, 'r', label = 'val_acc')
plt.plot(acc, 'b', label = 'acc')
plt.legend()
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
```

Out[89]: Text(0.5, 0, 'Epochs')

<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

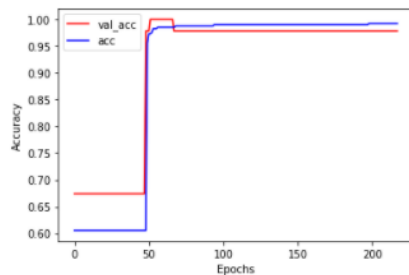


Figure 62. M#3: Loss and Accuracy of the Training Set and Validation Set

3.6.4 Loss and Accuracy of the Testing Set

I then calculate and print the loss and accuracy of the testing data.

```
In [90]: # Calculating the loss and accuracy of data tested
yts_cat = to_categorical(yts)
loss, acc = model.evaluate(xts, yts_cat)
print("Test Loss: ", loss)
print("Test Accuracy: ", acc)

114/114 [=====] - 0s 61us/step
Test Loss: 0.09127392845326349
Test Accuracy: 0.9736841917037964
```

Figure 63. M#3: Loss and Accuracy of the Testing Set

3.6.5 AUC Score and ROC Curve of Testing Set

I also find the AUC Score and ROC Curve for the testing and training data respectively.

```
In [91]: # Calculating the AUC score of Testing data
yts_pred = model.predict_proba(xts)
fal_pos_rate, tru_pos_rate, thresh = roc_curve(yts, yts_pred[:,1])
auc_krs = auc(fal_pos_rate, tru_pos_rate)
print('Testing data AUC: ', auc_krs)

Testing data AUC: 0.9927667269439421
```

Figure 64. M#3: AUC Score of Testing Set

```
In [92]: # Plotting the ROC curve of Testing data
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fal_pos_rate, tru_pos_rate, label = 'Keras (area = {:.3f})'.format(auc_krs))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc = 'best')
plt.show()
```

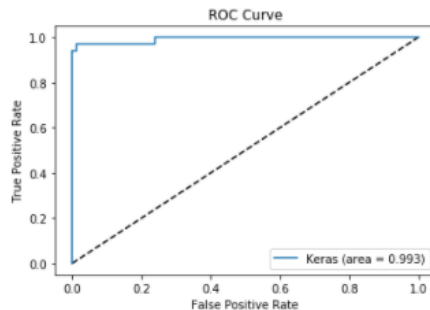


Figure 65. M#3: ROC Curve of Testing Set

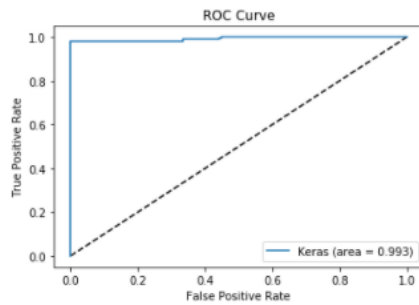
3.6.6 AUC Score and ROC Curve of Training Set

```
In [93]: # Calculating the AUC score of Training data
ytr_pred = model.predict_proba(xtr)
fal_pos_rate, tru_pos_rate, thresh = roc_curve(ytr, ytr_pred[:,1])
auc_krs = auc(fal_pos_rate, tru_pos_rate)
print('Testing data AUC: ', auc_krs)
```

Testing data AUC: 0.9930305755395684

Figure 66. M#3: AUC Score of Training Set

```
In [94]: # Plotting the ROC curve of Training data
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fal_pos_rate, tru_pos_rate, label = 'Keras (area = {:.3f})'.format(auc_krs))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc = 'best')
plt.show()
```



```
In [58]: ##### THE END #####
```

Figure 67. M#3: ROC Curve of Training Set

3.7 Conclusion

These were my three Neural Network Models: Logistic Regression, Softmax Regression and Deep Learning Softmax Regression Model. I also provided a comprehensive comparison between the performances of these three models. This comparison showed that the techniques were similar and highly accurate. The similarity may be a result of many factors, among which are the possible homogeneity of the data, the overall difficulty of the task, the parameter of the algorithms, etc. There are many advanced techniques data scientists use to optimize the performance of neural networks. These may be: Batch Normalization, Dropout and Weight Regularization, etc. Maybe by using one of these techniques, the performance might have had a slight improvement. Overall, we could see by extensive simulations on the chosen dataset that these procedures give satisfactory performances, although it should be noted that their results serve as supplementary information to the diagnostician and are not used as the final decision. For this essential reason, machine learning techniques are continuously being exploited in healthcare and clinical trial decision-making.

References

1. [Retrieved from Machine Learning Mastery website]:
<https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/>
2. [Retrieved from Stack Overflow website]:
<https://stackoverflow.com/questions/48845354/why-is-validation-accuracy-higher-than-training-accuracy-when-applying-data-augm>
3. [Retrieved from Machine Curve website]:
<https://www.machinecurve.com/index.php/2019/10/15/leaky-relu-improving-traditional-relu/>
4. [Retrieved from Machine Learning Mastery website]:
<https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>
5. [Retrieved from Deep AI website]:
<https://deeptai.org/machine-learning-glossary-and-terms/sigmoid-function>
6. [Retrieved from UCI Machine Learning Repository website]:
<https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29>
7. [Retrieved from Machine Curve website]:
<https://www.machinecurve.com/index.php/2020/02/21/how-to-predict-new-samples-with-your-keras-model/>
8. [Retrieved from Machine Learning Mastery website]:
<https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/>
9. [Retrieved from Medium website]:
<https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e>
10. [Retrieved from Machine Learning Cheatsheet website]:
[https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html#:~:text=delta\(x\)%20!-.ELU,to%20RELU%20except%20negative%20inputs](https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html#:~:text=delta(x)%20!-.ELU,to%20RELU%20except%20negative%20inputs)
11. [Retrieved from Machine Learning Mastery website]:
<https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/>

12. [Retrieved from Programmer Sought website]:
<https://www.programmersought.com/article/9830137163/>
13. [Retrieved from Medium website]:
<https://medium.com/analytics-vidhya/derivative-of-log-loss-function-for-logistic-regression-9b832f025c2d>
14. [Retrieved from Machine Learning Mastery website]:
<https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/>
15. [Retrieved from Keras website]:
<https://keras.io/>
16. [Retrieved from Kite website]:
https://www.kite.com/python/docs/keras.utils.to_categorical
17. [Retrieved from Wikipedia website]:
https://en.wikipedia.org/wiki/Softmax_function
18. [Retrieved from Wikipedia website]:
https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg
19. [Retrieved from Wikipedia website]:
https://en.wikipedia.org/wiki/Sigmoid_function
20. [Retrieved from Stack Overflow website]:
<https://stackoverflow.com/questions/48845354/why-is-validation-accuracy-higher-than-training-accuracy-when-applying-data-augm>
21. [Retrieved from Folsom Cordova Unified School District website]:
<https://www.fcusd.org/cms/lib/CA01001934/Centricity/Domain/664/Histology%20Lab%20-%20Revised%202016-2017.docx>
22. [Retrieved from Handspeak website]:
<https://www.handspeak.com/word/list/index.php?abc=tu&id=369>
23. [Retrieved from Github website]:
<https://github.com/keras-team/keras/issues/6104>